SEA-AC: Symbolic Execution-based Analysis towards Approximate Computing

Sara Ayman Metwalli
Tokyo Institute of Technology
sara@cad.ict.e.titech.ac.jp

Yuko Hara-Azumi
Tokyo Institute of Technology
hara@cad.ict.e.titech.ac.jp

ABSTRACT

Recently, the quest to reduce energy consumption in digital systems has been the subject of a number of ongoing studies. One of the most researched focus points is "Approximate Computing". Approximate computing is a new computing paradigm in both hardware and software designs that aim to achieve energy-efficient digital systems by applying relaxations to the designs and allowing them to have errors within an acceptable range. Although approximate computing is a broad and current interest of many researchers all over the world, the main question, "In which section of a program or a circuit approximations can be applied¿' has not been well answered yet. There exist some statistical works that tried to answer that question using trial-and-error approaches. This paper proposes a new approach to answer the above-mentioned question by using symbolic execution to eliminate the data-dependency and the time wastage of existing methods. Experiments on the proposed approach has shown up to 85%, in addition to eliminating data dependency and the need of the approximate version of the application.

KEYWORDS

Approximate computing, Analysis, Symbolic execution

ACM Reference Format:
Sara Ayman Metwalli and Yuko Hara-Azumi. 2018. SEA-AC: Symbolic Execution-based Analysis towards Approximate Computing. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO51).* ACM, New York, NY, USA, 2 pages.

Although AC is the focus of many researchers work, the answer to the question "In which section of a program or a circuit approximations can be applied" has not been well answered yet. Existing approaches [4] [5] [6] have attempted to answer this question using statistical approaches that require both the precise (original) and approximated versions of the application code and require a large amount of test data in order to verify whether any approximations are applicable or not. Every time an approximation is applied, the verification process needs to be repeated in order to be conducted, which makes these approaches quite time consuming, Our work addresses the above issue of analysis time and data dependency burdens, by developing a software framework SEA-AC (Symbolic Execution based Significance Analysis for Approximate Computing). SEA-AC analyzes the target application's precise code and guides the designer towards the program section(s) to which approximation can or cannot be applied. Differently than aforementioned trial-and-error-based statistical analysis, which requires both precise and approximated versions of the code and a massive amount of test data, SEA-AC statically analyzes the significance of variables in the precise version only and thus needs no specific test data nor any input from the designer. SEA-AC conducts the analysis through executing the application code symbolically using an SEE (Symbolic Execution Engine). The SEE used in this work is called KLEE [2]. The proposed framework result in a ranking of the independent variables/ inputs based on how much they effect the quality of the output, an overview of the processes conducted by SEA-AC in Figure 1. Whilst symbolic execution has several benefits, it still faces some limitations, mainly path explosion and environment interactions. The number of executable paths in a program generally can grow rapidly as the program size increases, and it might even become infinite in such cases where the program has unbounded loops. This, however, can be avoided by path-finding increasing code coverage, in addition to reducing execution time through merging similar paths [1]. Moreover, program slicing (as a pre-processing step) can also be used to solve this problem.

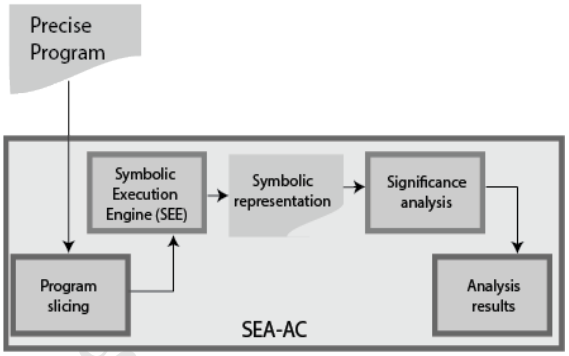

Figure 1: The outline of SEA-AC

To compute the significance of each independent input/variable, SEA-AC calculates three different weights for each independent input/variable:

- **Coefficient-based weight**: The coefficient-based weight is calculated in SEA-AC using a method called Linear Regression Coefficients(LRC). LRCs are estimates of the significance of the effect in a dependant quantity when applying changes on its independent factors (or, to use the language more commonly associated with regression, the independent variables and the dependent variable). This method expresses the weight of a variable as a normalized value, based on the

Table 1: Weights for the operation-based analysis.

Significance	Operation type
Highest	Multiplication, shift left
Medium high	Addition, subtraction
Medium low	Division, shift left
Lowest	Logic operations

coefficient of the variable against those of all variables. That is, assume that the j^{th} variable in path p is 'x', the weight of 'x', $W_{LRC}(x,p)$, as shown in Eqn (1):

$$W_{LRC}(x,p) = \frac{B_j}{\sum_{j'}^{J} B_{j'}} \quad (1)$$

- **Operation-based weight**: The operation-based weight considers the type of operations conducted between the independent inputs/variables. Each operation (e.g., summation, multiplication, shift operations, etc) has a different degree of effect on the output as described in Table 1, meaning that some operations may have a big effect on the quality of output while others may not have that much of an effect on the output. SEA-AC assigns different weights to the independent inputs/variables based on those operations and how much they affect the output. Let's consider the following equation 'z = 3 + 2×5 -3 + 6/2'. The levels of operations included control how this calculation should be done. Since the priority in arithmetic operations is in the order of multiplication/division and, then addition/subtraction, the value of 'z' in this example will be 13. SEA-AC follows the same level importance distribution when computing the operation-based weight of independent variables/inputs.
- **Occurrence-based weight**: Any application has several possible outcomes depending on the input data. The occurrence-based weight considers the appearance of a specific independent input/variable in all possible outcomes of the application, assigning a higher weight to the inputs/ variable that appears in more possible outcomes than others.For example, in case of two outputs, 'w = x + 2y' and 'z = 4x', the occurrence of 'x' and 'y' is 2 and 1, respectively (i.e., the weights of variables 'x' and 'y': $W_{OCB}(x,p)$=2 and $W_{OCB}(y,p)$=1). That is to say, to calculate the occurrence-based weight, we sum up the number of appearances of individual variable in each of the program outputs. It should be noted that, in some cases, a variable would appear more than once in an expression, such as 'z = 2x + 4y + x/y', where further simplification cannot be done. This case affects the LRC-based analysis, not the occurrence-based analysis – for the variable 'x', LRC-based weight ($W_{LRC}(x,p)$) is 3/7 (= (2+1)/(2+4+1)), while occurrence-based weight ($W_{OCB}(x,p)$) is 1.

After calculating the above-explained weights, and in order to generate the ranking of independent variables/ inputs based on their significance SEA-AC computes two types of overall weights for each independent input/variable as follows.

(1) **Path-based Total Weight (W_{PB})**: The path-based or output-based significance calculates the significance of a variable for a specific path/ output. In this case, the occurrence-based weight is omitted because it only considers the occurrence of variables in different paths/ outputs as shown in Eqn (2). As shown in the equation, p stands for the probability of which a path is to occur. This depends on the application field and the user preferences.

$$W_{PB}(x,p) = W_{LRC}(x,p) + W_{OPB}(x,p) \quad (2)$$

(2) **Overall Total Weight (W_O)**: Overall total weight calculates the average significance of each variable considering all the outputs of the program as shown in Eqn (3).

$$W_O(x) = \sum_{p} P_p \times W_{PB}(x,p) \quad (3)$$

The path possibility P_p can be determined by either of (1) the user's designation, (2) the profiles obtained in advance, or (3) range analysis on branch variables [3].

Experimental results for six types of DSP applications show that SEA-AC can successfully extract the significance ranking of independent inputs/variables similarly as to which that can be obtained from existing statistical works, while taking much less analysis time (on average 88% and up to 97% reduction). These results indicate that SEA-AC can simplify the decision of which section(s) of the target code can approximated or must be precise, compared with statistical approaches that are inevitably data-dependent and time consuming. It should be noted that SEA-AC can handle not only software designs but also hardware designs, especially because hardware has been designed at higher abstraction level these days. Our work will be of great help to optimize hardware designs with approximate computing techniques, in terms of area, power/time consumption, and/or performance. Especially because we are now approaching the end of Moore's law, approximate computing is a key enabling technology for both hardware and software. Future extensions of SEA-AC include allowing the optimization and the application of approximate computing techniques on the cross-layer of hardware and software levels in order to reach the best approximation possible and hence leading to the best possible reduction of power/energy consumption.

ACKNOWLEDGMENTS

This work is in part supported by JSPS KAKENHI 17H04677.

REFERENCES

[1] P. Boonstoppel, C. Cadar, and D. Engler. 2008. RWset: Attacking path explosion in constraint-based test generation. In *Proc. of Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems*. 351–366.
[2] C. Cadar et al. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs.. In *OSDI*, Vol. 8. 209–224.
[3] M. Gort and J. Anderson. 2013. Range and bitmask analysis for hardware optimization in high-level synthesis. In *Proc. of Asia and South Pacific Design Automation Conference*. 773–779.
[4] A. K. Mishra, R. Barik, and S. Paul. 2014. iACT: A software-hardware framework for understanding the scope of approximate computing. In *Proc. of Workshop on Approximate Computing Across the System Stack*.
[5] R. Venkatesan et al. 2011. MACACO: Modeling and analysis of circuits for approximate computing. In *Proc. of Computer-Aided Design*. 667–673.
[6] Q. Zhang et al. 2014. Approxit: An approximate computing framework for iterative methods. In *Proc. of Design Automation Conference*. 1–6.